TrimBot2020 Deliverable 5.2

Scene Flow Software

Principal Author:	ALUF
Contributors:	ALUF
Dissemination:	PU

Abstract: A major goal in WP5 is an accurate scene flow estimation software that can run at interactive framerates. In this report, we describe our solution for a scene flow module comprising disparity and optical flow estimation.

Deliverable due: Month 24

Contents

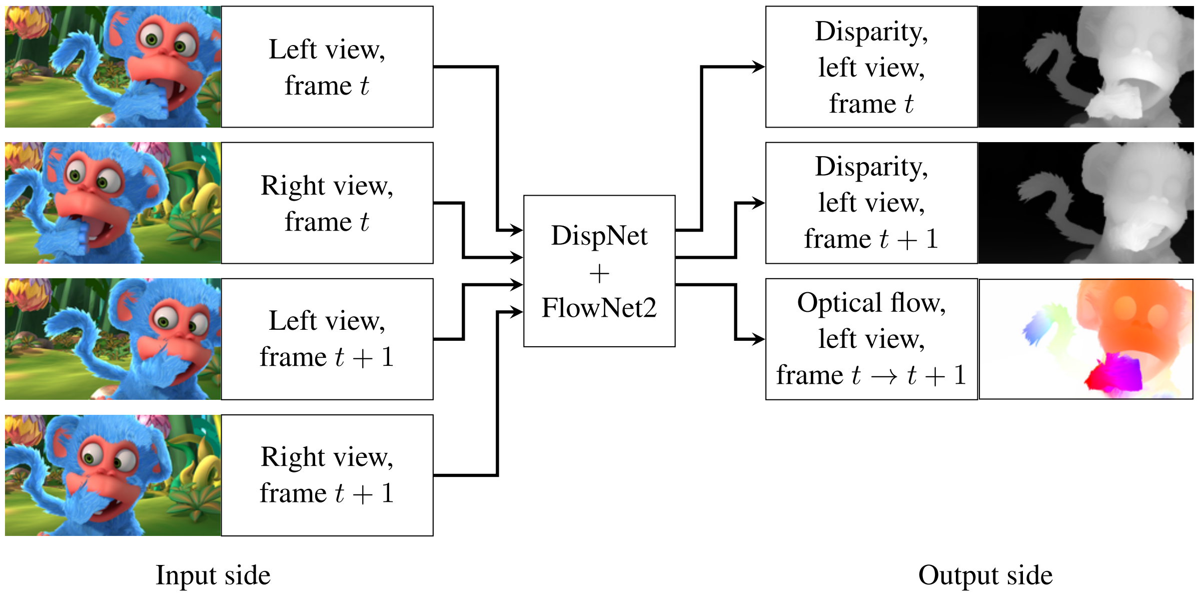


Figure 1: **Required data** (input side) and **produced data** (output side) of the scene flow module. The module takes streams of camera images and produces streams of scene flow data. The raw output formats (disparity, optical flow) decouple the module from the intrinsic camera parameters and the calibration of the stereo rig (which cannot be assumed to be both perfectly constant and known beforehand). With known camera intrinsics and stereo baseline, the raw outputs are then easily converted to metric scene flow. Images taken from [1].

1 Scene Flow

Scene flow describes the 3D motion of an observed scene. This motion can be induced by any combination of movements of the scene objects or the camera itself. Our approach to scene flow combines depth estimation and optical flow estimation as illustrated in Fig. 1. By estimating the depth of a scene point in a given camera frame, this point is located within 3D space. Optical flow is then used to track the point into the next frame where a new depth estimate is computed. These three components together yield a 3D motion vector for the observed point.

For depth estimation, we use the **DispNet** from [1], a convolutional neural network (CNN) for full-frame disparity estimation on rectified stereo images. This is a *disparity* method, but given the intrinsic and stereo-rig calibration data of the stereo camera, the disparity results are equivalent to metric depth estimates.

The **FlowNet2** CNN from [2] is used to estimate the required optical flow maps. As with the DispNet, the FlowNet2's results are given in pixel space. Again, camera calibration data is necessary to lift the result into metric scene space.

2 Software Package

The ALUF scene flow package is provided within the ROS ecosystem and split into multiple ROS nodes:

- **`aluf_ethzcam_rectifier`**: Undistortion and stereo-rectification of raw camera image streams (camera calibration data must be given)
- **`aluf_dispnet, aluf_disparity_view`**: DispNet module for disparity estimation, along with a visualization helper module so the disparity outputs can be visually inspected
- **`aluf_flownet, aluf_flow_view`**: FlowNet2 module for optical flow estimation, along with a visualization helper
- **`aluf_vision_pipeline`**: A meta-package including a ROS launchfile to start all scene flow components
- **`aluf_test_images_repeater`**: Contains a ROS node which repeatedly publishes two included stereo frames (= 4 images). This package can be used for testing whether the other scene flow nodes work without having to connect and calibrate a camera.

The DispNet and FlowNet2 modules are both built upon the Caffe deep learning framework [3]. The CNN definitions and models are modular. New trained models or definitions can easily be swapped for the existing ones; this is akin to changing (hyper)parameters in a “traditional” algorithm. Each ROS node is a separate ROS package so the disparity and optical flow estimators can also be used each on their own.

2.1 ROS Interface

Our rectification node subscribes to the two raw images stream published by the camera driver. The camera calibration is done beforehand (we used Kalibr [4]) and given to the rectifier as a `YAML` text file. The rectifier publishes two new image streams and a `camera_info` stream for the left-view image stream. The `camera_info` messages contain the recified camera intrinsics. We also misuse the (normally unused) projection matrix field `P` of the message and annotate it with additional information about the rectified images; currently `P` tells which areas of the images can be cropped away as they contain undistortion artifacts, and it contains the stereo baseline of the camera system. The exact use of `P` is documented in the rectification node’s source code[1].

2.2 DispNet

The DispNet module subscribes to the synchronized left and right camera topics. For each processed frame pair, the module produces a disparity map which is published as a topic of `DisparityImage` messages (the optional disparity visualization module subscribes to this

[1]File path in the project repository tree, at time of writing: `Proc3D/SceneFlowALUF/ROS-packages/aluf_ethzcam_rectifier/rectify.py`

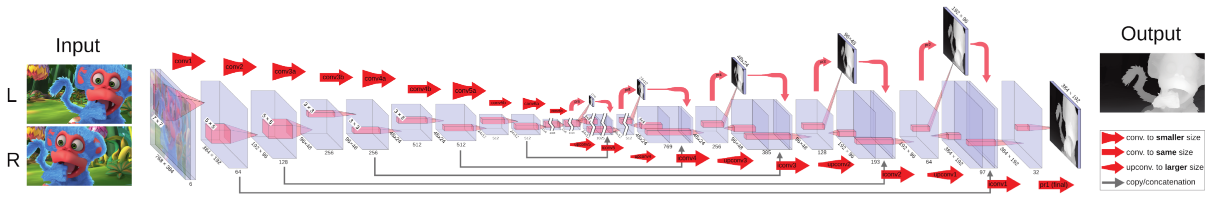


Figure 2: **DispNet architecture.** See [1] and the document for project deliverable D5.1 for details. This is the "simple" architecture without a correlation layer.

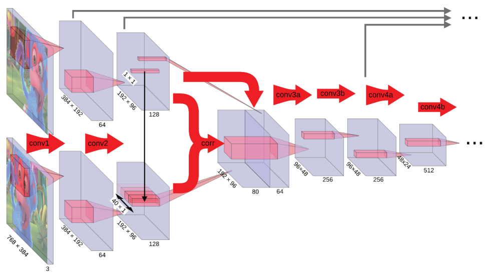


Figure 3: **DispNet architecture with correlation layer.** This architecture is the basis for the DispNetCorr1D from [1]. The part shown here replaces the early stages of the DispNet in Fig. 2.

topic and itself publishes another `Image` topic with a displayable colormapped version of the disparity map). The DispNet itself directly produces a full-frame disparity map, but some postprocessing options are implemented and can be enabled and configured using parameters in the DispNet module's launchfile. Fig. 4 shows examples for the following options:

- If **left-right consistency** checking is enabled, the node will not only produce a disparity map for the left view, but also for the right view. Both maps are then used to check if the disparity estimates are consistent across views (i.e. in both left-right and right-left directions). Pixels with inconsistent estimates are flagged. Note that this has a significant performance penalty as the compute hardware effectively has to do twice as much work.
- Disparity estimates for pixels at the far left of the left view are subject to occlusion effects: the DispNet will estimate a disparity for a pixel even if that pixel is not actually visible in the right view. An optional filtering step flags such **"impossible" matches**. Note that this cannot detect general occlusions arising from the perspective difference between the two camera views.
- The DispNet tends to produce oversmoothed disparity maps, i.e. the map is incorrectly smoothed where depth discontinuities should be (e.g. along object boundaries). An optional filtering step flags areas with **large disparity gradients**.

2.3 FlowNet2

The FlowNet module takes images from the left camera as input and outputs optical flow fields. The parameter `flownet_variant` determines which variant is being used and has

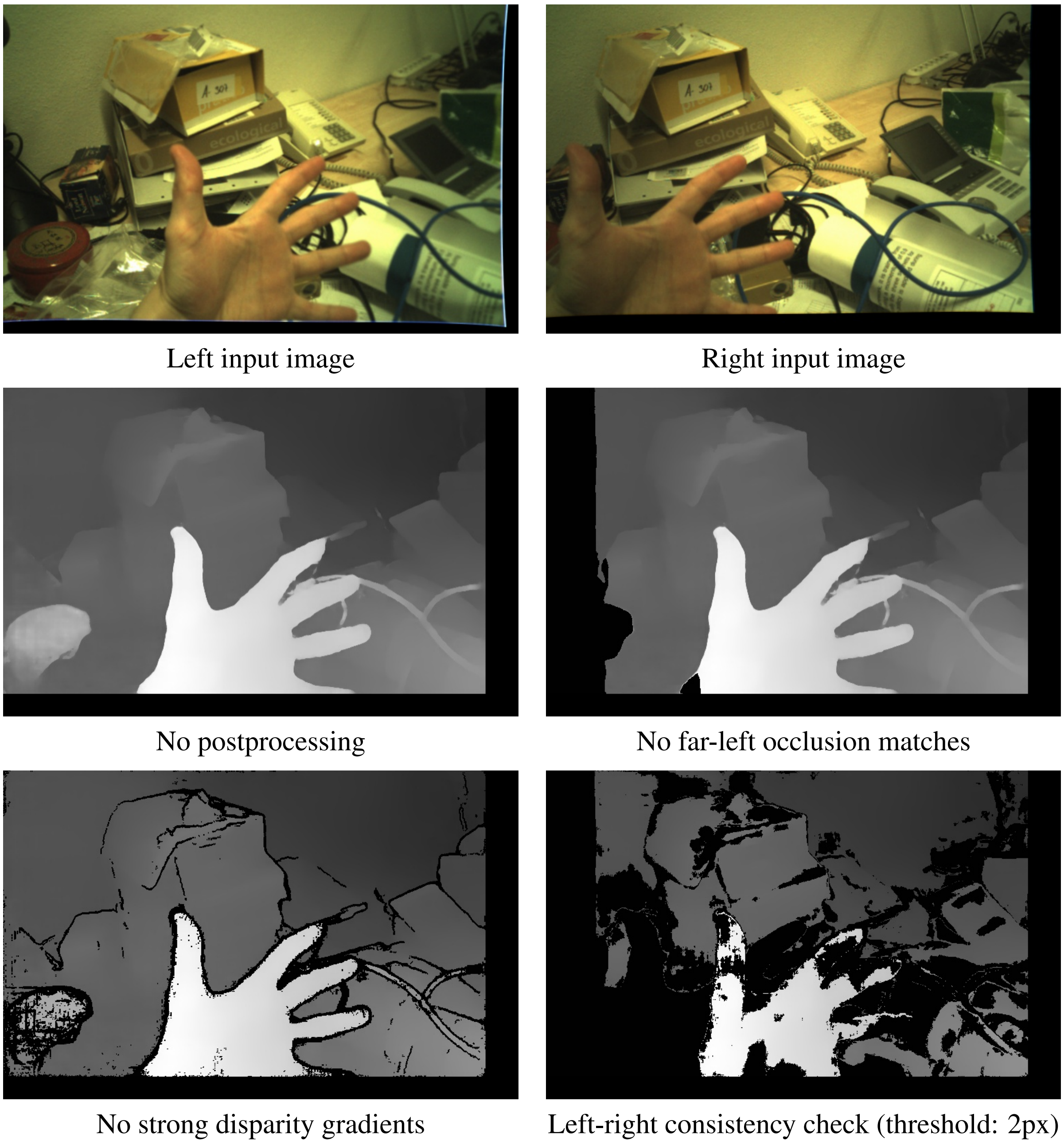

Left input image
Right input image
No postprocessing
No far-left occlusion matches
No strong disparity gradients
Left-right consistency check (threshold: 2px)

Figure 4: Disparity results on different postprocessing configurations. Note that left-right consistency checking eliminates the area occluded by the hand as well as featureless regions in which accurate matching is not possible.

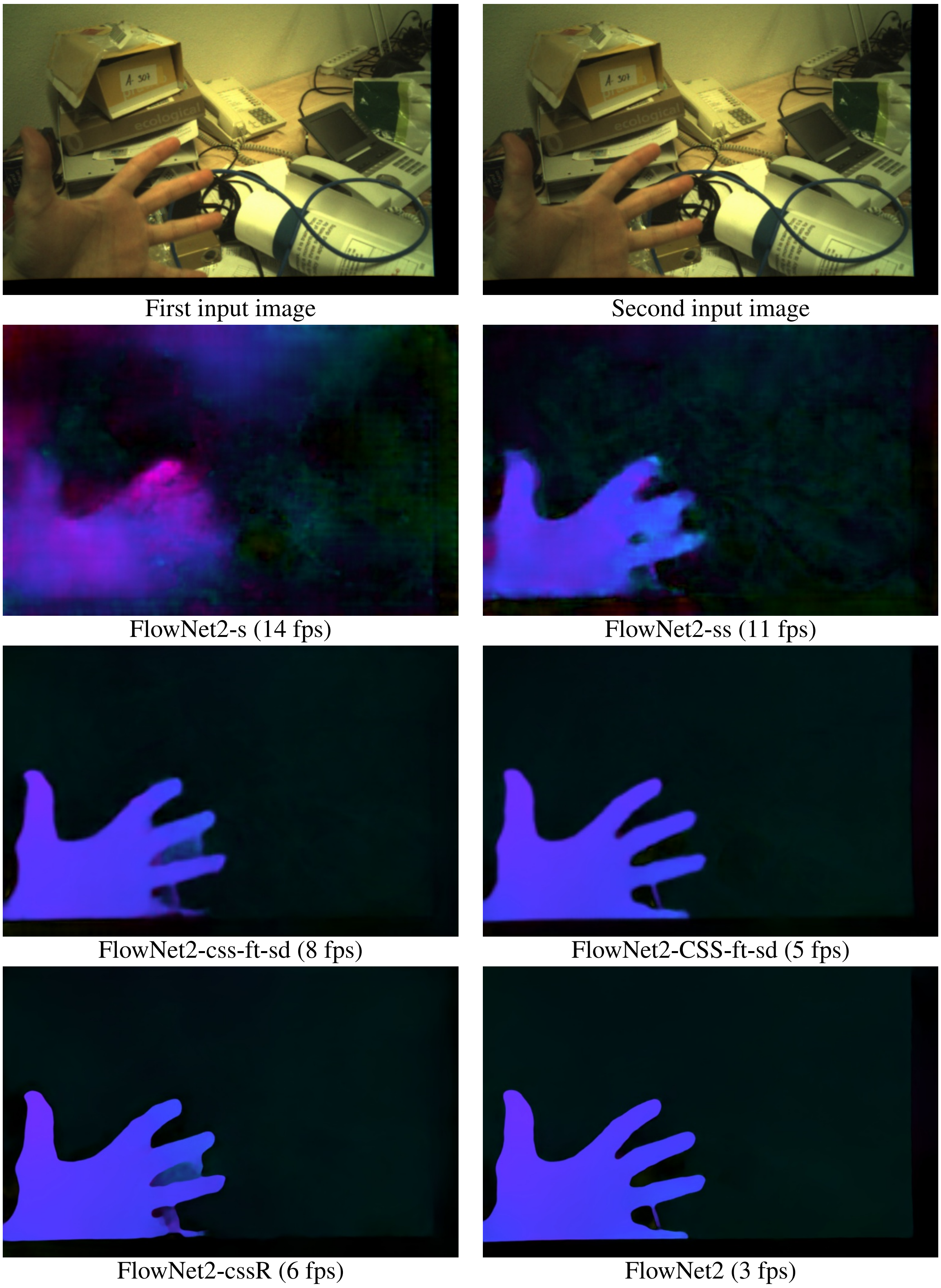


Figure 5: Optical flow results on different FlowNet2 architecture configurations. The flow fields are visualized using the "Middlebury" scheme shown in Fig. 7.

the following options, sorted in order from low accuracy/low runtime to high accuracy/high runtime (see [2] for details). Fig. 5 shows an example for each configuration. Fig. 6 shows schematics of the different architectures.

- **FlowNet2-s**: This is the smallest and fastest FlowNet (only containing a FlowNetS). Accordingly, the accuracy is also the lowest.
- **FlowNet2-ss**: This is a stack of two small FlowNetS. The resulting network yields more accurate estimates than FlowNet2-s, but less accurate than the other, larger architectures.
- **FlowNet2-css-ft-sd**: This is a stack of one small FlowNetC and two small FlowNetS. This stack gives the best performance and runtime trade-off. This variant is also finetuned on small dislacement data, which is often present in real-world scenarios.
- **FlowNet2-cssR**: This is the same network as above, but includes a final refinement network for high resolution refinement (with the architecture of the fusion network from FlowNet2, but without FlowNet2-SD). This produces sharper boundaries and less noise than FlowNet2-css-ft-sd.
- **FlowNet2-CSS-ft-sd**: This is the same as FlowNet2-css-ft-sd but with a much larger network.
- **FlowNet2**: This is the complete FlowNet2 (slowest and best performance).

The optical flow is output through an `Image` sensor message, where the image contains floating point vectors with x- and y-components for each pixel. The visualization module can be used to convert this into an interpretable image. Flow is usually visualized by placing the flow vector in a color circle. Color then indicates direction and brightness indicates magnitude. There are two types of visualization (Parameter `viz_type`):

- **Middlebury Vizualization** (value `middlebury`): Best used for computer screens, the center of the color circle is black (i.e. black indicates zero motion, see Figure 7a).
- **Sintel Vizualization** (value `sintel`): Best used for printing, the center of the color circle is white (i.e. white indicates zero motion, see Figure 7b).

Since different applications have different typical flow ranges, there is a scale parameter for the visualization (Parameter `viz_scale`). The meaning of its value is chosen such that it corresponds to the flow magnitude at maximum saturation (choose appropriately for small or large flows).

2.4 Framerates And Resource Usage

These numbers were evaluated on a machine with a Nvidia GTX 980M mobile GPU with 8 GB VRAM. In terms of performance, this GPU is comparable to the Nvidia GTX 1060 model with 6 GB VRAM chosen for the project hardware.

Module	Configuration	VRAM use	Framerate
DispNet	DispNetCorr1D	1 GB	12 fps
FlowNet2	FlowNet2-s	0.4 GB	14 fps
	FlowNet2-ss	0.6 GB	11 fps
	FlowNet2-css-ft-sd	0.9 GB	8 fps
	FlowNet2-CSS-ft-sd	1.6 GB	5 fps
	FlowNet2-cssR	1.6 GB	6 fps
	FlowNet2	3.0 GB	3 fps

2.5 Launch Example

Note that the `aluf_vision_pipeline` package launchfile combines steps 4, 6, 7, 9, and 10.

```
# 0.) Compile software
catkin_make install

# 1.) Plug in camera

# 2.) Set permissions of camera device
sudo chmod o+rwX /dev/ttyACM0

# 3.) Launch camera driver
roslaunch uvc_ros_driver uvc_ros_driver.launch
# This provides the following topics:
#   /uvc_camera/cam_1/image_raw
#   /uvc_camera/cam_0/image_raw
#

# 4.) Launch the rectifier:
roslaunch aluf_ethzcam_rectifier rectify-alufcam.launch
# This maps the following topics:
#   /uvc_camera/cam_1/image_raw  ->  /uvc_cam1_rect_mono
#                                    /uvc_cam1_rect_mono/camera_info
#   /uvc_camera/cam_0/image_raw  ->  /uvc_cam0_rect_mono
#                                    /uvc_cam0_rect_mono/camera_info

# 5.) Launch image viewers
rosrun image_view image_view image:=/uvc_cam1_rect_mono
rosrun image_view image_view image:=/uvc_cam0_rect_mono
```

```
# 6.) Launch DispNet
roslaunch aluf_dispnet dispnet.launch
# This publishes the /dispnet topic

# 7.) Launch disparity visualization
roslaunch aluf_disparity_view dispview.launch
# This subscribes to /dispnet and publishes /dispnet_vis

# 8.) Launch a viewer for disparity visualization
rosrun image_view image_view image:=/dispnet_vis

# 9.) Launch FlowNet
roslaunch aluf_flownet flownet.launch
# This publishes the /flownet topic

# 10.) Launch flow visualization
roslaunch aluf_flow_view flowview.launch
# This subscribes to /flownet and publishes /flownet_vis

# 11.) Launch a viewer for flow visualization
rosrun image_view image_view image:=/flownet_vis
```

References

[1] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation," in *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. [Online]. Available: `http://lmb.informatik.uni-freiburg.de/Publications/2016/MIFDB16`.

[2] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. [Online]. Available: `http://lmb.informatik.uni-freiburg.de/Publications/2017/IMSKDB17`.

[3] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22nd ACM international conference on Multimedia*, ACM, 2014, pp. 675–678.

[4] P. Furgale, J. Rehder, and R. Siegwart, "Unified temporal and spatial calibration for multi-sensor systems," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, IEEE, 2013, pp. 1280–1286.

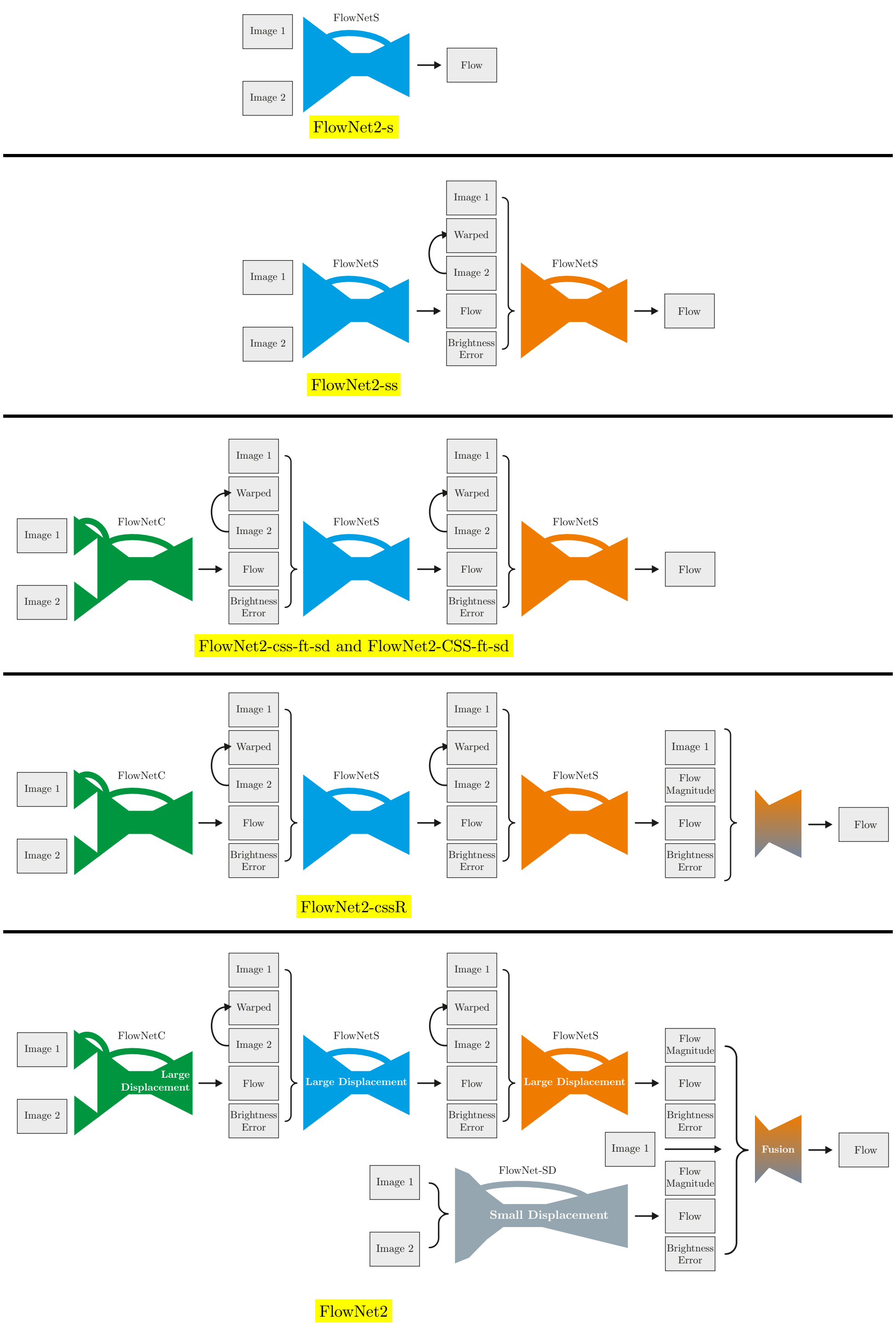
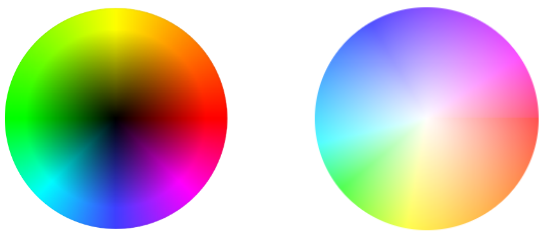


Figure 6: **FlowNet2 architectures.** Configurations of the FlowNet2 for the networks listed in Section 2.3. Lowercase letter c/s designate that a network uses fewer channels per layer than the corresponding uppercase (C/S) network.



(a) Middlebury (b) Sintel

Figure 7: Color circles used for Middlebury and Sintel optical flow visualization. To apply the visualization the optical flow vector is placed in the circle; a zero flow vector is black in Middlebury but white in Sintel. In both styles, larger flows appear as more saturated colors. The Sintel style is more printer-friendly while a Middlebury flow display is often easier for humans to interpret.